
Divorcing Language Dependencies from a Scientific Software Library

Gary Kumfert,

with

Scott Kohn, Jeff Painter, & Cal Ribbens

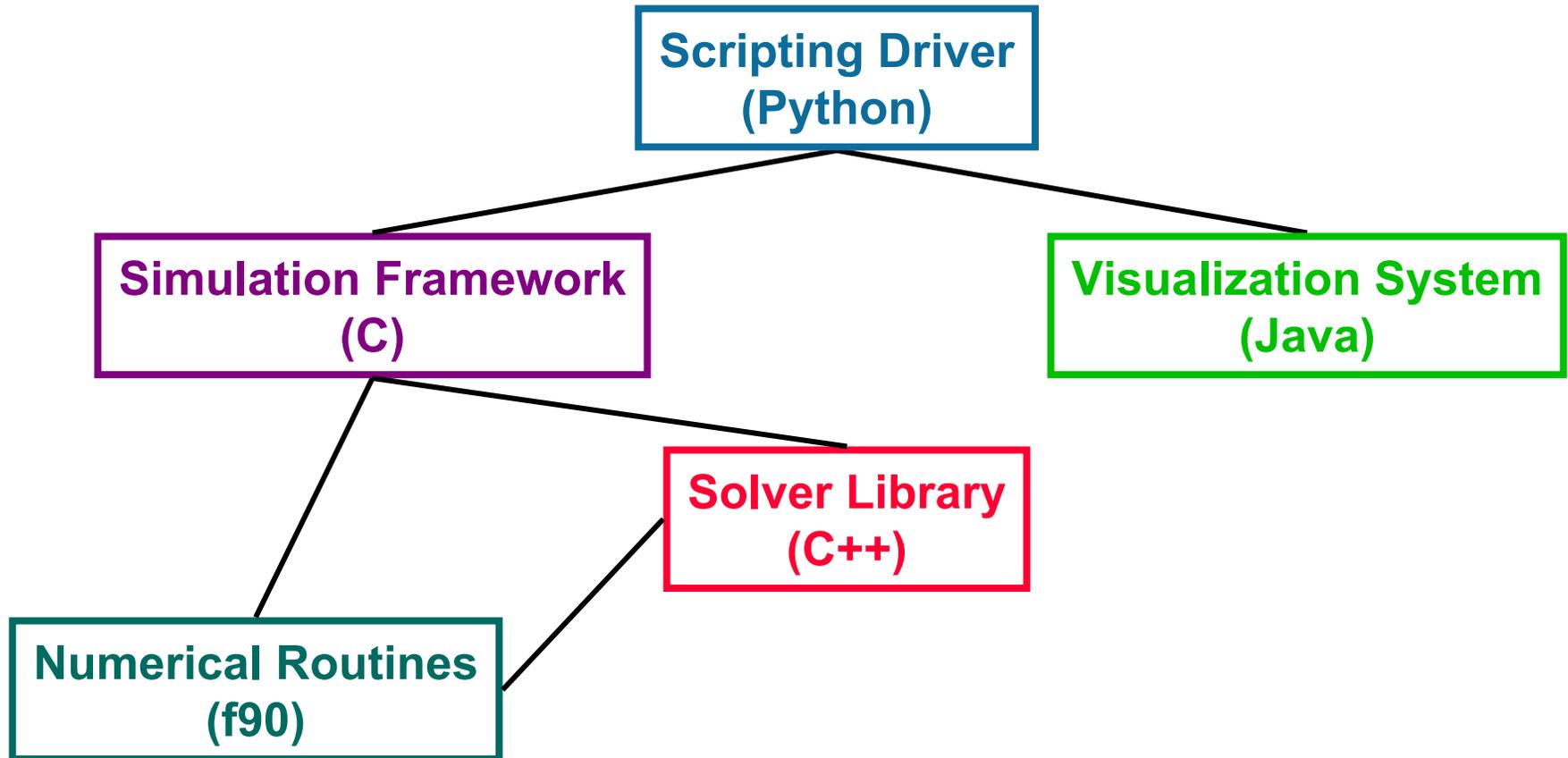
LLNL VaTech



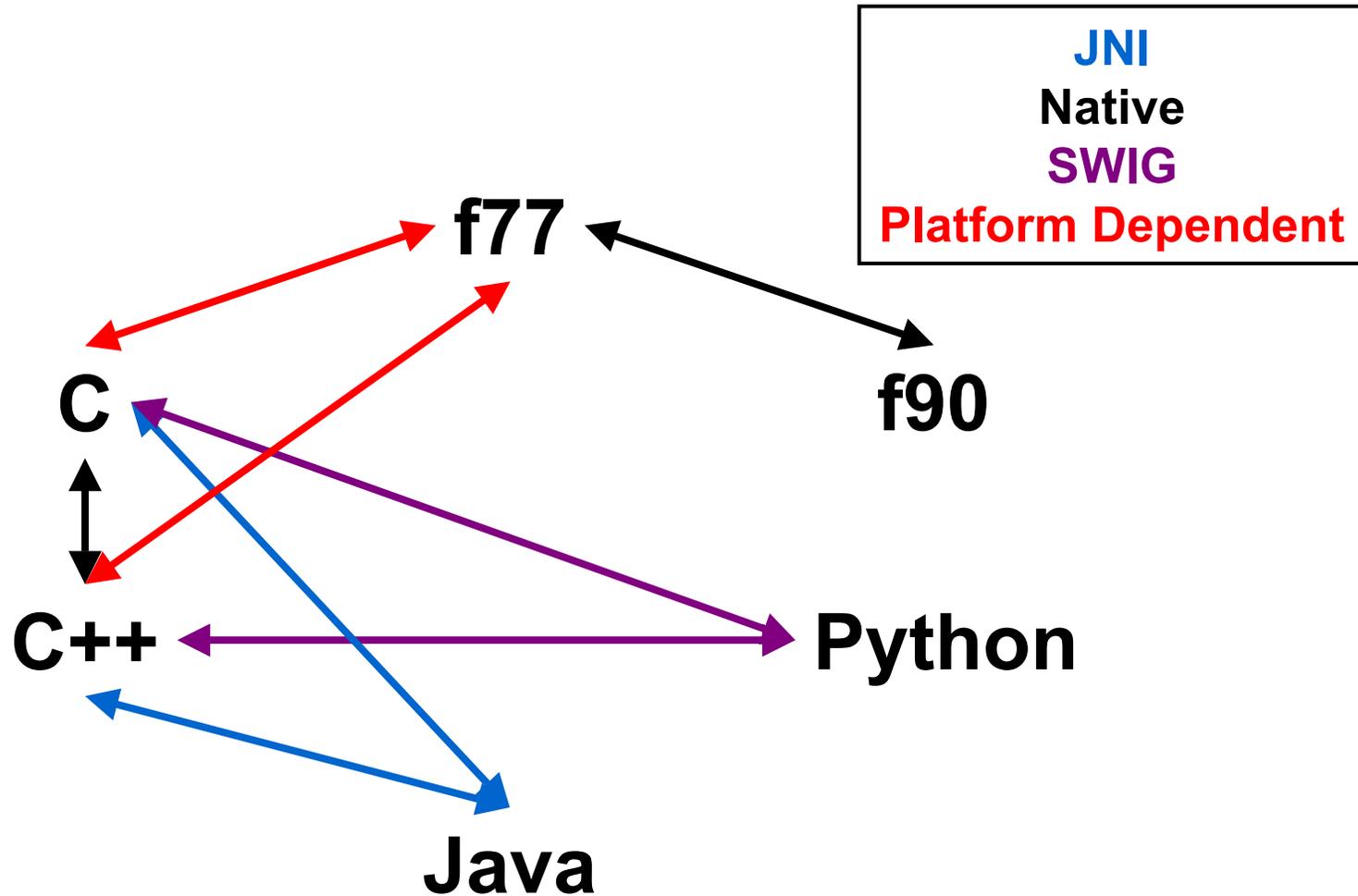
BABEL

- **Language Interoperability Tool**
 - ◆ You specify “interfaces” in our language
 - ◆ We generate glue code between application and library
- **Part of a Component Framework**
 - ◆ Enables OOP in non-OOP languages
 - ◆ Enables safe Dynamic Casting and QueryInterface capabilities

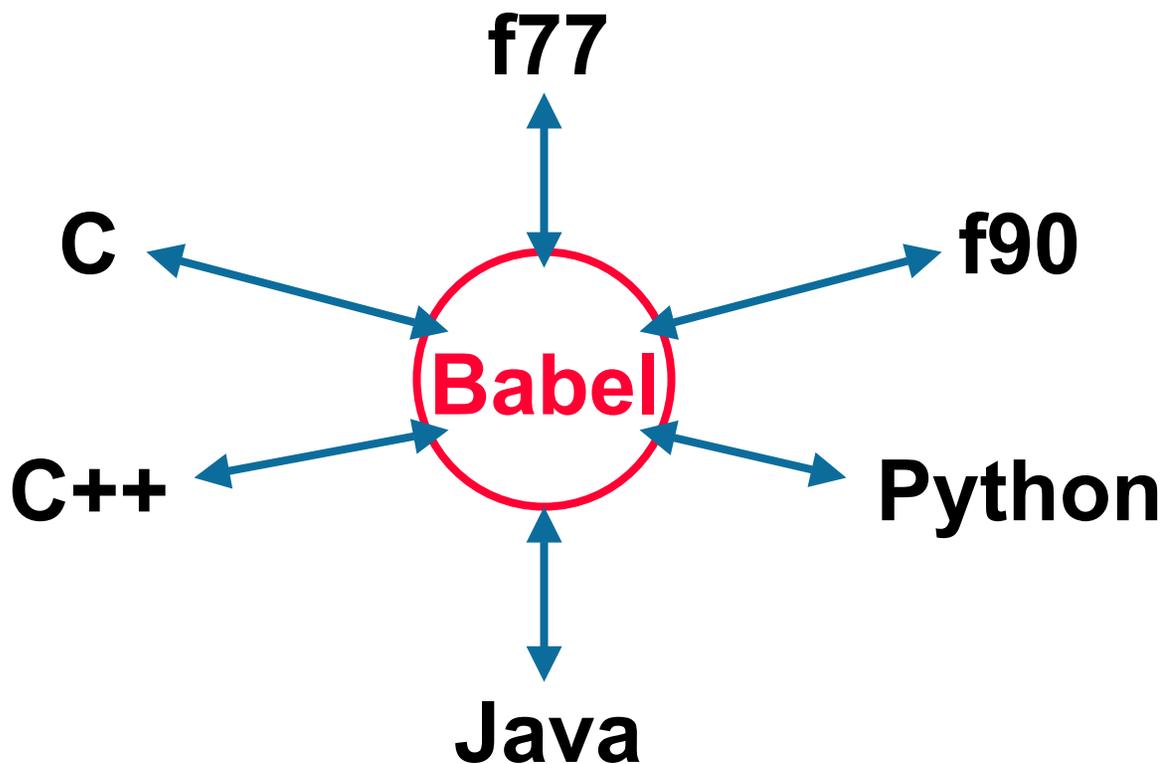
What I mean by “Language Interoperability”



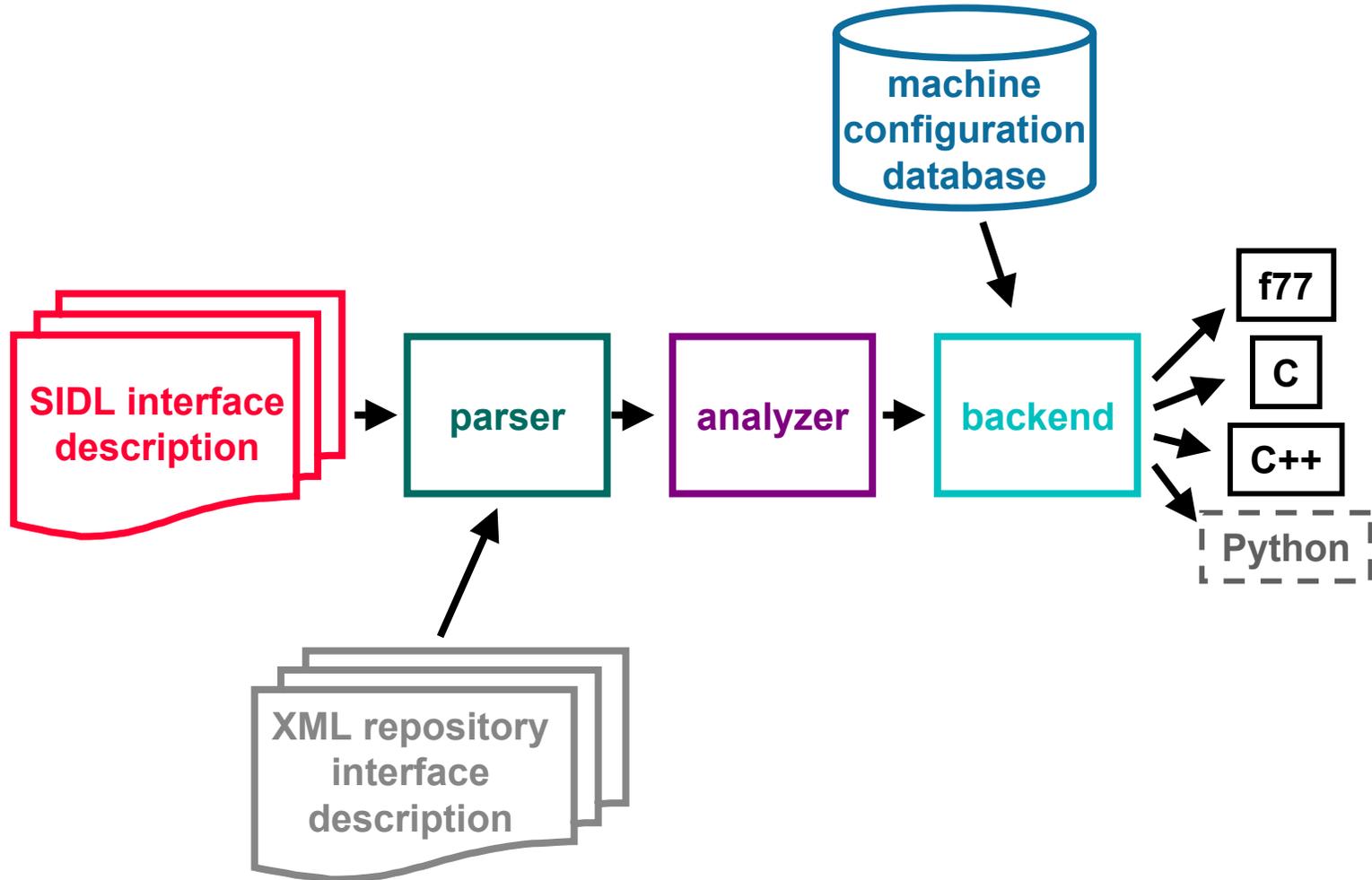
Hand Coded Solutions



Objects, Virtual Functions, RMI & Reference Counting: all from Babel



Babel generates glue code



Scientific Interface Definition Language (SIDL)

```
version Hypre 0.5;
version ESI 1.0;

import ESI;

package Hypre {
    interface Vector extends ESI.Vector {
        double dot(in Vector y);
        void axpy(in double a, in Vector y);
    };
    interface Matrix {
        void apply(out Vector Ax, in Vector x);
    };
    class SparseMatrix implements Matrix, RowAddressable {
        void apply(out Vector Ax, in Vector x);
    };
};
```

class
exception
interface
package

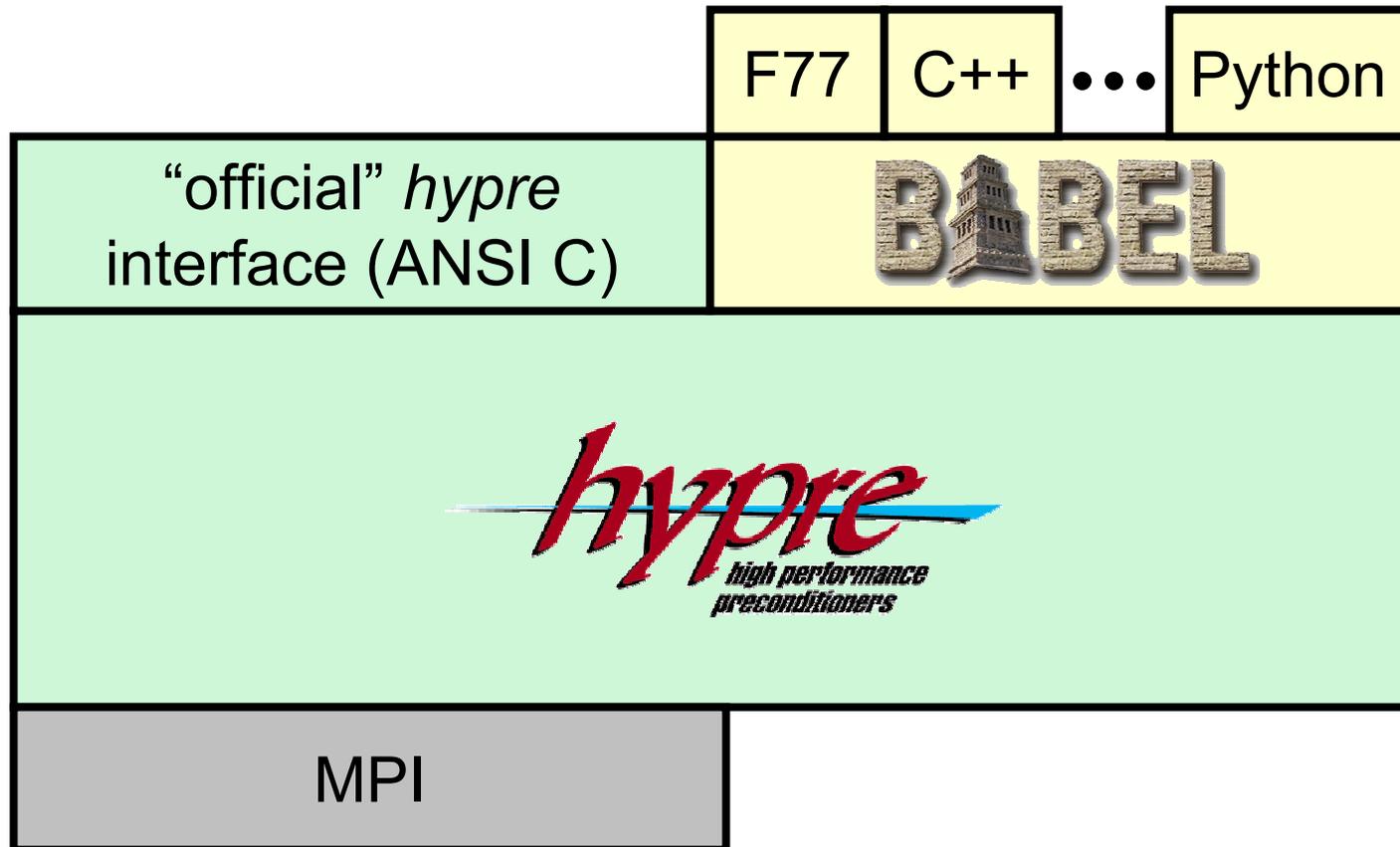


-
- **Software to be “divorced” from its language dependence**
 - **Scalable parallel linear solvers and preconditioners (LLNL)**
 - **Implemented in ANSI C using MPI**
 - **“Object Based”**

Collaboration Objectives

- **Babel side:**
 - ◆ **demonstrate Babel technology**
 - ◆ **feedback from library developers**
- **Hypre side:**
 - ◆ **Automatically create Fortran bindings**
 - ◆ **Explore new designs**
 - ▶ **Object-Oriented**
 - ▶ **Component-Based**
 - ◆ **Integrate other software**
 - ▶ **C++ or F77**

Envisioned Architecture



Approach

- Identify minimal working subset of *hypre*
 - ◆ Structured Solvers
- Create SIDL description
- Add base classes to create heirarchy
- Tie generated code to existing *hypre* library
- Iterate

Problem: Creating wrong types

- SIDL has 3 types of objects
 - ◆ interfaces - no implementations (pure abstract)
 - ◆ abstract classes - partial implementations
 - ◆ concrete classes - full implementations
- Users were creating abstract classes when they meant to create concrete classes

```
interface Foo {
    int doThis( in int i );
    int doThat( in int i );
}

class Bar implements Foo {
    int doThis( in int i );
};

class Grille implements Foo {
    int doThis( in int i );
    int doThat( in int i );
};
```

Solution: Fix The Grammar

- Added the “**abstract**” keyword
 - ◆ Compiler issues error if a method is undefined and class is not declared abstract
- Added the “**implements-all**” keyword
 - ◆ declares all methods as overridden
 - ◆ saves user typing

```
interface Foo {
    int doThis( in int i );
    int doThat( in int i );
}

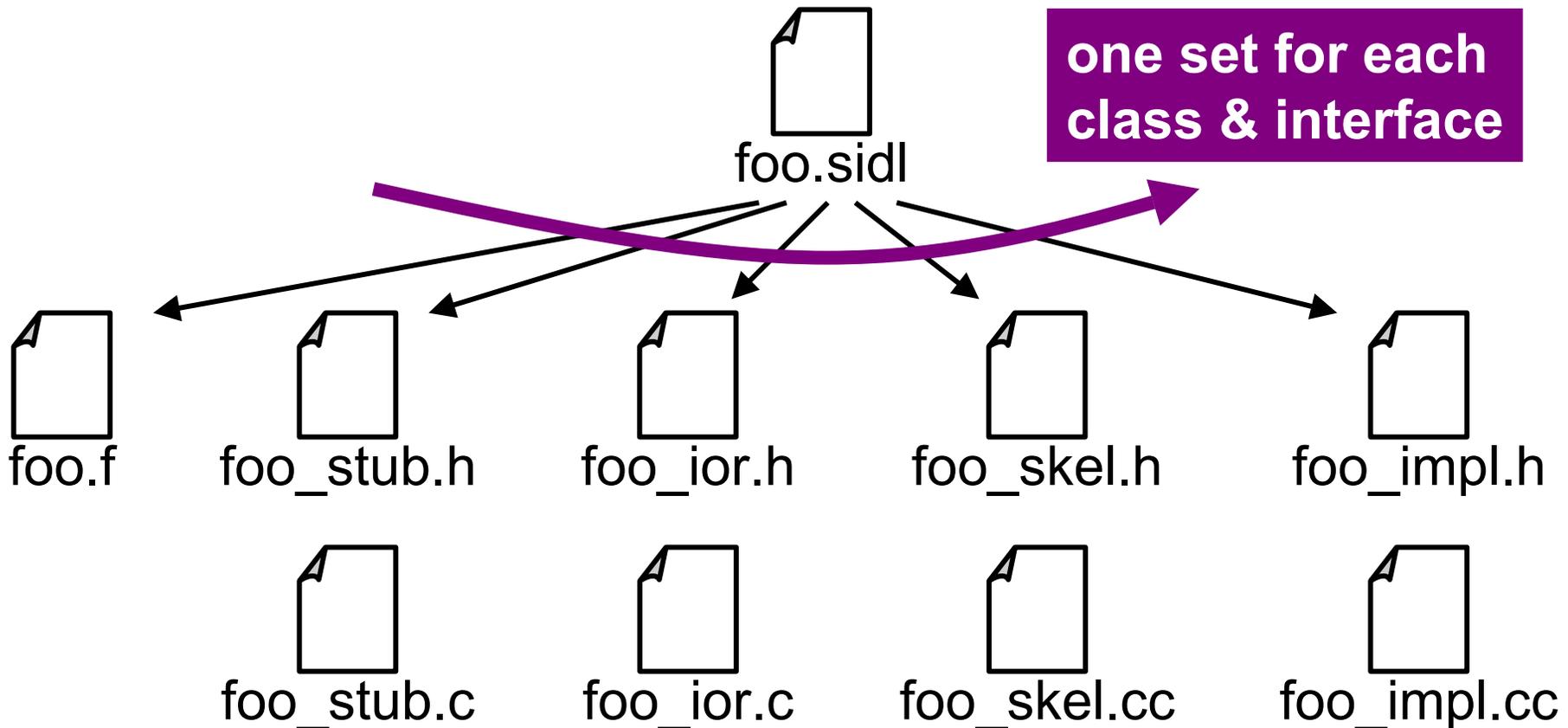
abstract class Bar implements Foo {
    int doThis( in int i );
};

{

};
```

Problem: Managing all the Files

- Babel creates many source files



Solution: Babel Generates Makefile Macros

- A “babel.make” file is generated

```
IORSRCS = foo_ior.c \  
         bar_ior.c \  
         grille_ior.c  
  
IORHDRS = foo_ior.h \  
         bar_ior.h \  
         grille_ior.h
```

- Users include it into their own makefiles
 - ◆ They control the build rules
 - ◆ We provide the file names

Problem: Incremental Development

- Library Developer would do the following:
 - ◆ write SIDL file
 - ◆ run Babel to generate bindings
 - ◆ hand edit “Impl” files to call their library code

```
#include "mylib.h"

int impl_Foo_doThis( Foo * self, const int i ) {

    return  mylib_Foo_doThis(
                (mylib_Foo*) self->userdata,
                i
            );

}
```

Problem: Incremental Development (2)

- Now assume this was done for 20 classes, each with 20 methods.
- Now assume a class needed a 21st method
- Babel would regenerate all files and wipe out Developer's edits

```
#include "mylib.h"

int impl_Foo_doThis( Foo * self, const int i ) {
    return mylib_Foo_doThis(
        (mylib_Foo*) self->userdata,
        i
    );
}
```

Solution: Code Splicing

- Added preservation of developer's edits
- Code Splicer works line-by-line
 - ◆ interleaves old code into new code
 - ◆ looks for begin-end pairs embedded in comments

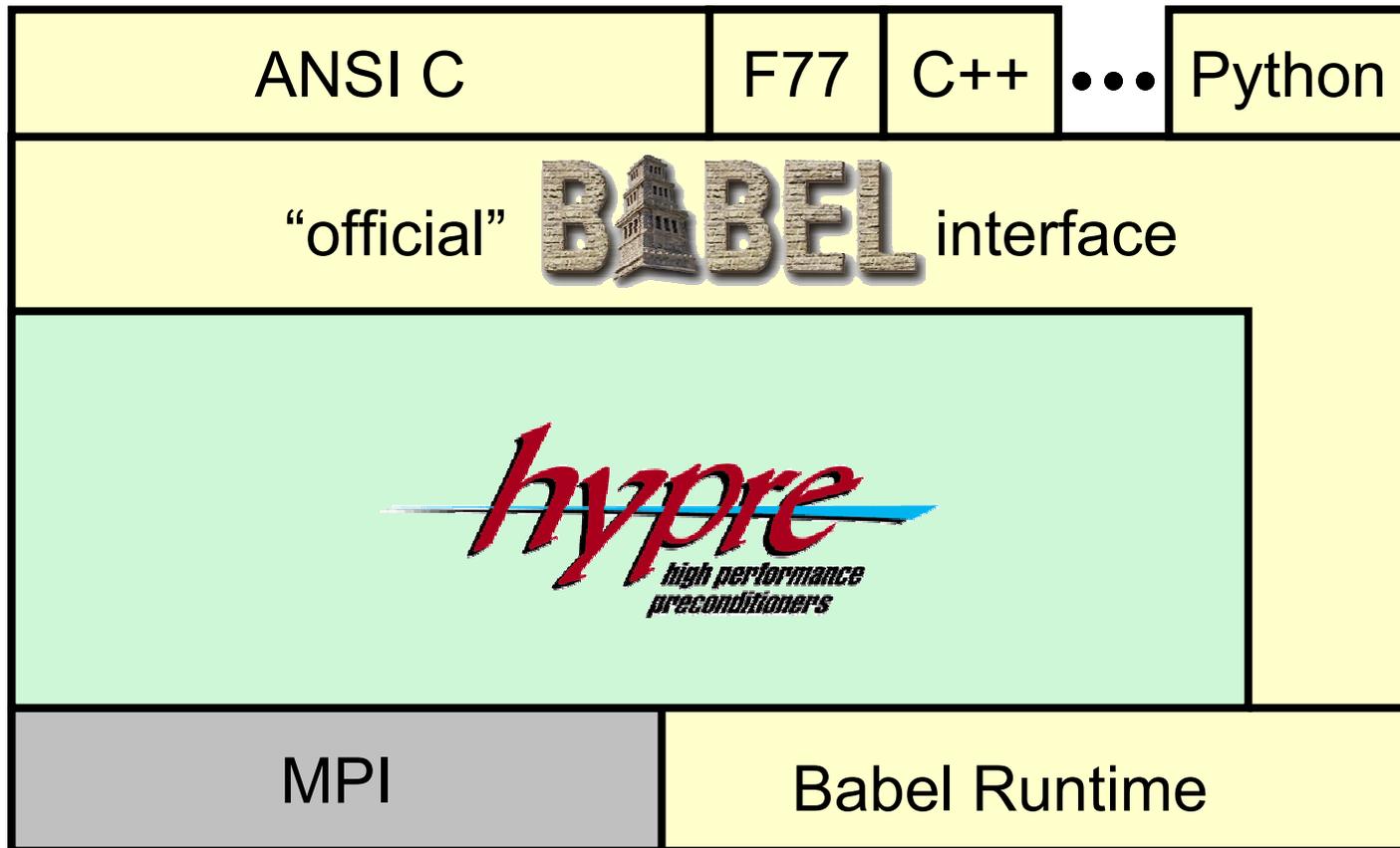
```
/* DO NOT DELETE splicer.begin( user-includes ) */
#include "mylib.h"
/* DO NOT DELETE splicer.end( user-includes ) */

int impl_Foo_doThis( Foo * self, const int i ) {
    /* DO NOT DELETE splicer.begin( Foo_doThis ) */
    return mylib_Foo_doThis(
        (mylib_Foo*) self->userdata,
        i
    );
    /* DO NOT DELETE splicer.end( Foo_doThis ) */
}
```

Results

- Call *hypre*
 - ◆ from C, F77, or C++
 - ◆ on SPARC Solaris or DEC/OSF
 - ◆ (more languages & platforms coming)
- No interference with MPI
- Babel overhead within runtime noise

Best Result: Change of Architecture



Reasons for Change

- Liked using the tool
- No Hand F77 bindings
 - ◆ incompatible
 - ◆ outdated
- Preferred discussing designs in SIDL
 - ◆ easy for email
 - ◆ impossible to mix implementation & interface
- Convinced of Babel's longevity
- Babel enforces regularity in code
- Liked automatic reference counting
- Excellent compromise between:
 - ◆ Wanting polymorphism and OO techniques
 - ◆ Wanting all ANSI C for maximum portability

Current & Future Work

- **Language Support**
 - ◆ **Current: C, C++, F77, Python (Client)**
 - ◆ **Coming: Python(Server), Java, F90, Matlab**
- **Platform Independence**
 - ◆ **Implies RMI / Distributed Computing**
 - ◆ **SOAP**
- **Parallel Data Redistribution**
- **Babelization efforts in LLNL**
 - ◆ *hypre*
 - ◆ **SAMRAI**
 - ◆ **ALPS**

Public Beta Release

Late Summer

Components **@lnl.gov**

- **Our Website**

<http://www.lnl.gov/CASC/components>

- ◆ **Alexandria (Component Repository)**
- ◆ **Quorum (Online Voting)**
- ◆ **Generic Parallel Redistribution**

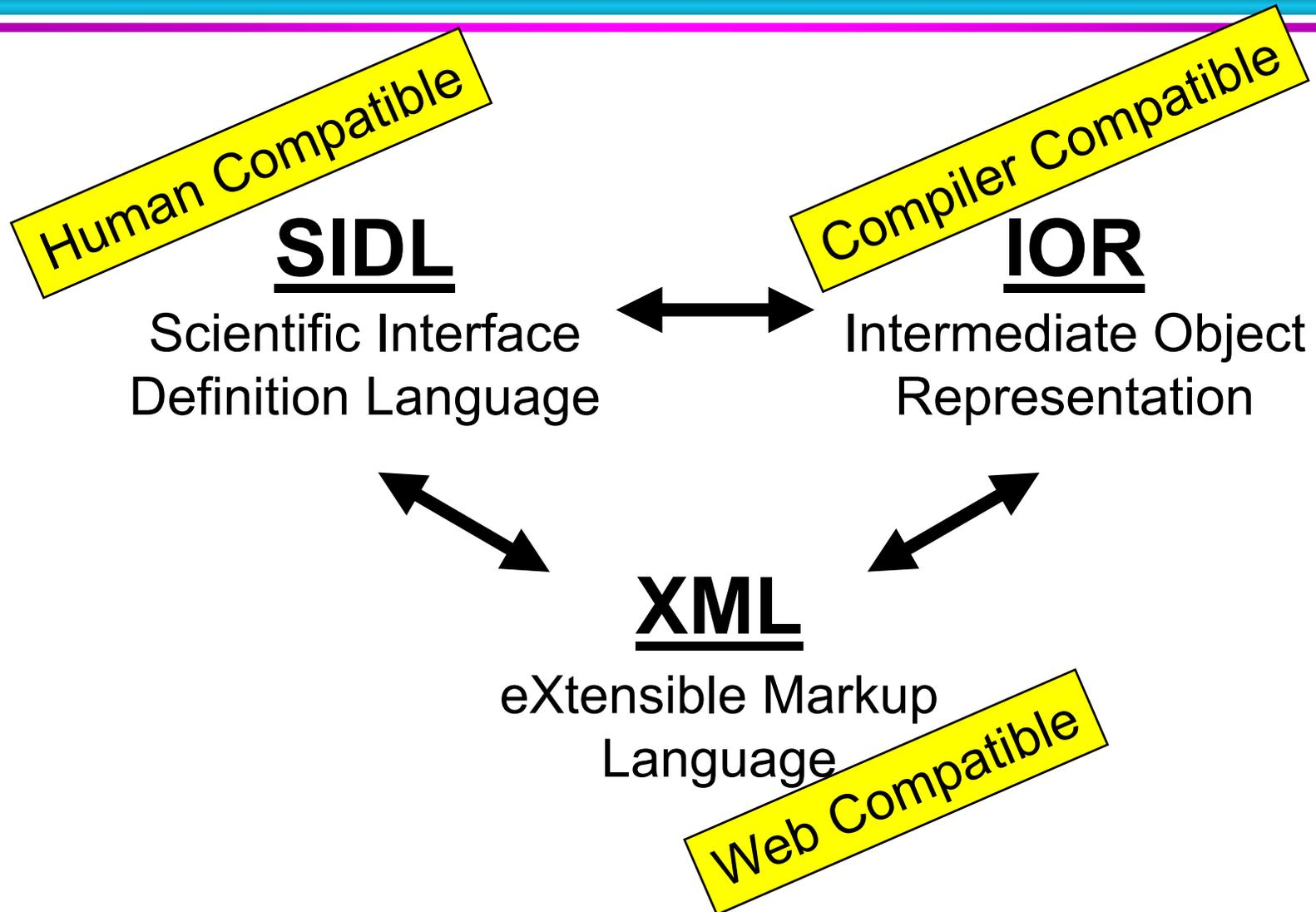
- **hypre**

<http://www.lnl.gov/CASC/hypre>

UCRL-VG-140349 Rev 1

Work performed under the auspices of the U. S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-Eng-48

Key to Babel's Interoperability...



Business Component Frameworks

- **CORBA**

- ☞ **Language Independent**
- ☞ **Wide Industry Acceptance**
- ☞ **Primarily Remoting Architecture**

- **Enterprise Java Beans (EJB)**

- ☞ **Platform Independent**
- ☞ **Runs wherever Java does**

- **COM**

- ☞ **Language Independent**
- ☞ **Most Established**
- ☞ **In Process Optimization**
- ☞ **Network Transparent**

Science

~~Business~~ Component Frameworks

- **CORBA**

- 📄 Language Independent
- 📄 Wide Industry Acceptance
- 📄 Primarily Remoting Architecture
- 🖱️ Huge Standard
- 🖱️ No In-Process Optimization

- **COM**

- 📄 Language Independent
- 📄 Most Established
- 📄 In Process Optimization
- 📄 Network Transparent
- 🖱️ not Microsoft Transparent
- 🖱️ Relies on sophisticated development tools

- **Enterprise Java Beans (EJB)**

- 📄 Platform Independent
- 📄 Runs wherever Java does
- 🖱️ Language Specific
- 🖱️ Potentially highest overhead

- **All The Above**

- 🖱️ No Complex Intrinsic Datatype
- 🖱️ No Dynamic Multidimensional Arrays
- 🖱️ No Fortran77/90/95 bindings
- 🖱️ No Parallel Components
- 🖱️ No Concept of SPMD Programming